Session 132

Graphics & Imaging Tips & Tricks

David Gelphman
Graphics and Imaging

Agenda

- Using Quartz 2D in Carbon applications on Mac OS X
- Carbon printing grafPort on Mac OS X
- Quartz performance tuning

Using Quartz 2D in Carbon Apps on Mac OS X

- Quartz has powerful 2D capabilities
- Consistent with PostScript imaging model
- Excellent replacement for PostScript PicComments
  - Images with all output devices, not just PostScript devices
- Assume some familiarity with Quartz 2D drawing model
- Reminder: Quartz is Mac OS X only, not CarbonLib

Demo

Quartz 2D Drawing in Carbon on Mac OS X

PDF File Import Is Easy!

```
url = CFURLCreateFromFSRef(NULL, &fsRef);
doc = CGPDFDocumentCreateWithURL(url );
bbox = CGPDFDocumentGetMediaBox(doc, 1);
CGContextDrawPDFDocument(ctx, bbox,
                                    doc, 1);
```

- Straightforward use of Quartz 2D drawing, even for QuickDraw based app

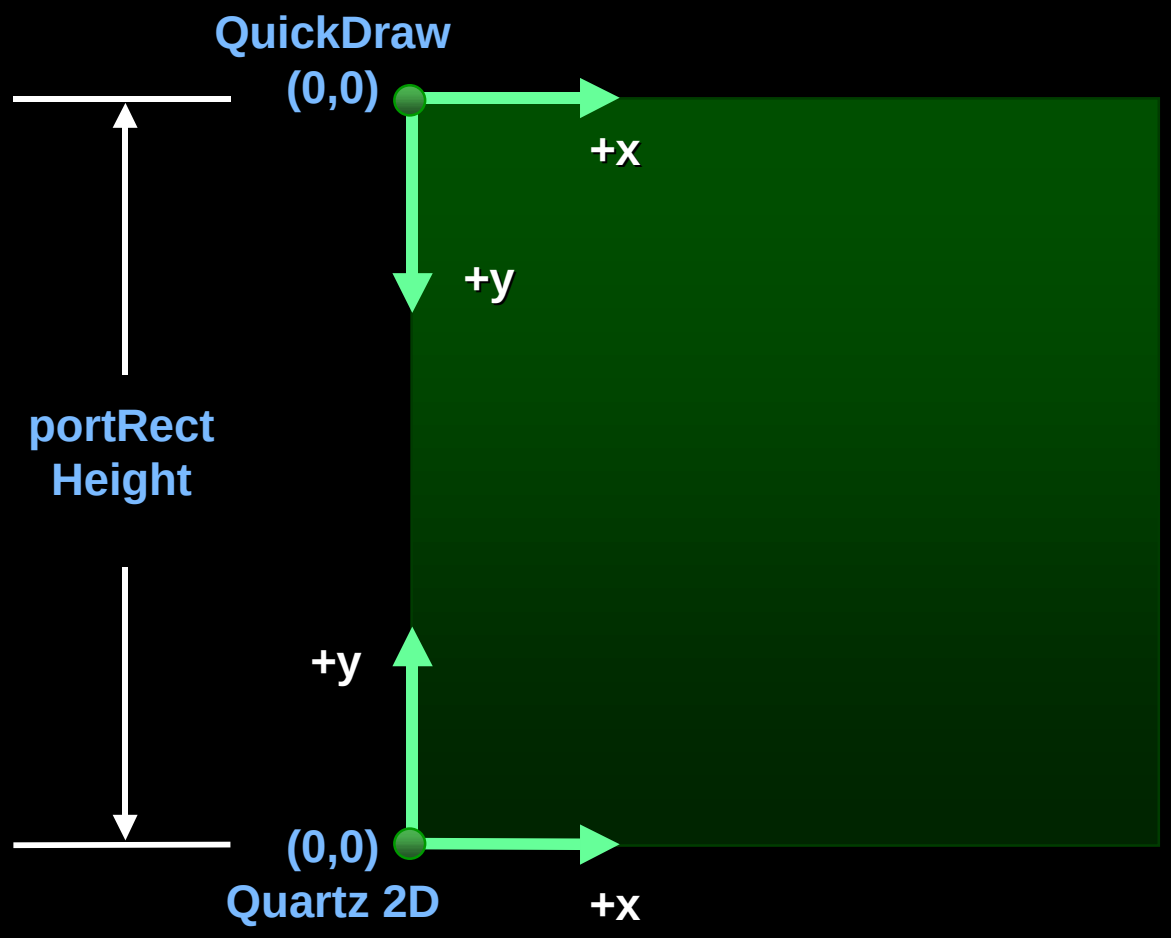
Using Quartz in Carbon on Mac OS X

- Obtain a CGContext (Quartz drawing port) from QuickDraw grafPort
- The graphics state in the CGContext is independent from grafPort graphics state
  - Fill and Stroke Color,
    Fill and Stroke ColorSpace
  - Clip
  - Line width
  - . . .

Quartz 2D Drawing Coordinate System

- Based on PostScript/PDF imaging models
- Standard Cartesian coordinates
  - +x goes across screen/page
  - +y goes up screen/page

QuickDraw and Quartz 2D Coordinate Systems



Coordinate System Issues

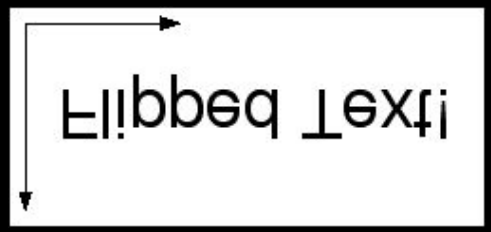
- Apps are used to QD coordinate system
  - Event coordinates returned in QD coordinate system
- Some apps want to continue to use QD style coordinate system
- Approach A: transform coordinates on the fly
  - **portBoundsHeight = portRect.bottom - portRect.top;**
  - **CGy = portBoundsHeight - QDy;**
  - Disadvantage: rotations and scaling of CGContext make calculations tricky

Coordinate System (Cont.)

- Approach B: use Quartz coordinate transforms

```
CGAffineTransform qdTransform =
CGAffineTransformMakeTranslation(
                    0, portBoundsHeight);
qdTransform =
   CGAffineTransformScale(qdTransform, 1, -1);
CGContextConcatCTM(ctx, qdTransform);
```

  - Disadvantage: causes text to be flipped!



Coordinate System (Cont.)

- Consider Approach B with modification
- Use Quartz coordinate transform but “unflip” coordinate system before and after drawing text
- Need to pass -y as text starting point

```
CGContextScaleCTM(ctx , 1, -1);
… text drawing using -y coordinates…
CGContextScaleCTM(ctx , 1, -1);
```

- Only needed if app wants Quartz 2D coordinate system to match QuickDraw coordinate system

Text Drawing Into CGContext

- Use of low level Quartz API for drawing text is discouraged
  - Not appropriate for drawing international text
- Use ATSUI or MLTE for drawing text into CGContext

ATSUI Drawing Into CGContext

```
ByteCount iSize = sizeof(CGContextRef);
ATSUAttributeTag iTag = kATSUCGContextTag;
ATSUAttributeValuePtr iValuePtr=&ctx;
ATSUSetLayoutControls( theLayout, 1,
        &iTag, &iSize, &iValuePtr );
...
ATSUDrawText(theLayout, ...);
```

QuickDraw CGContext Routines

- CreateCGContextForPort
  - Create a new CGContext from a port
- SyncCGContextOriginWithPort
  - Intializes CTM on context (removes scaling, etc.)
  - Resets CGContext origin so Quartz 2D origin has correct relationship to QuickDraw origin
- ClipCGContextToRegion
  - Resets clip on CGContext to that supplied

Screen Drawing Issues

- Window resizing
  - Sync CGContext with port clip
- Scrolling or port origin changes
  - Sync CGContext with port origin
- What about printing?

Printing Issues: Types of Carbon Applications

- Applications that use only Quartz 2D when printing
  - Application content imaged with Quartz 2D
  - Legacy PICT data? Coming soon: QDPictDrawToCGContext
  - Other types of QD content to draw to CGContext?
- QuickDraw only applications
- Applications that mix QuickDraw imaging with Quartz 2D imaging when printing

Printing on Mac OS X

- Printing architecture on Mac OS X is completely different than Mac OS 8/9
- Printing system leverages Quartz 2D drawing and PDF
- All printing generates a PDF spool file
- Same printing port for raster and PostScript printing
- High-quality rendering on all printing devices

Only Quartz 2D Imaging During Printing

- Before calling PMSessionBeginDocument, notify printing system

```
CFStringRef          s[1];
CFArrayRef    graphicsContextsArray;
s[0] = kPMGraphicsContextCoreGraphics;
graphicsContextsArray = CFArrayCreate(NULL, s, 1,
                                      &kCFTypeArrayCallBacks);
PMSessionSetDocumentFormatGeneration(printSession,
                    kPMDocumentFormatPDF,
                     graphicsContextsArray, NULL);
CFRelease(graphicsContextsArray);
```

Only Quartz 2D Imaging During Printing

- After each PMSessionBeginPage call, obtain the CGContext

```
CGContextRef printingContext = kPMNoReference;
err = PMSessionGetGraphicsContext(printSession,
        kPMGraphicsContextCoreGraphics,
        (void **)&printingContext );
```

- Quartz drawing origin is lower left corner of sheet, *not* imageable area

QuickDraw Printing grafPort on Mac OS X

- Apps using any QuickDraw imaging when printing use the QuickDraw printing grafPort
- After calling PMSessionBeginPage, obtain printing port

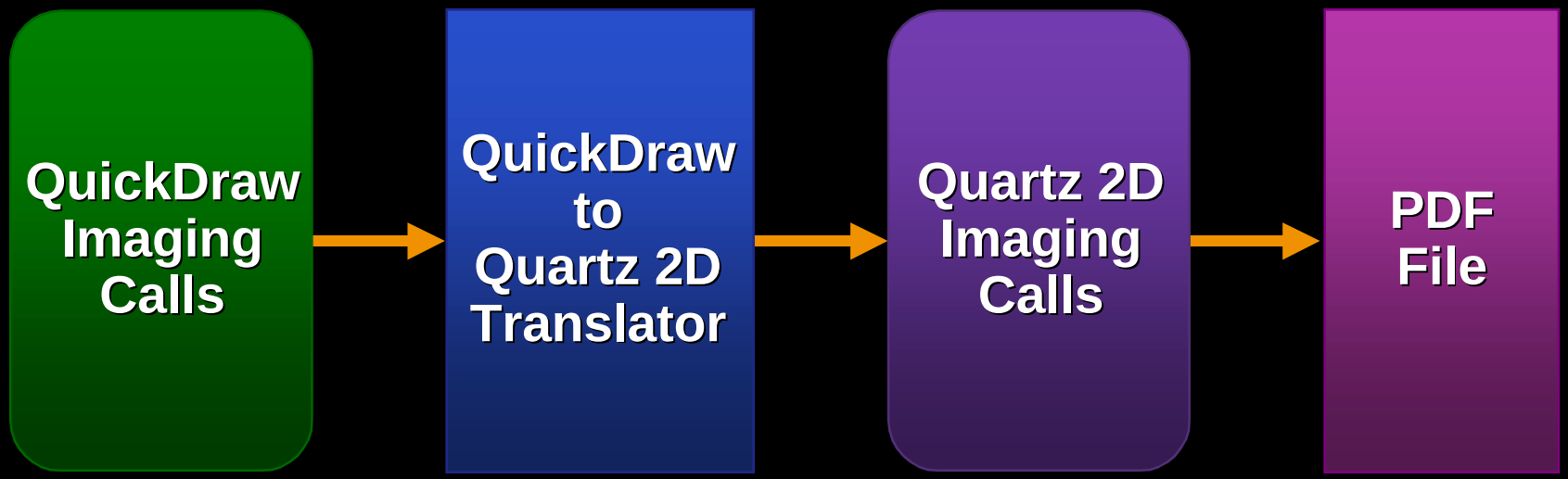
```
CGrafPtr printingPort = kPMNoReference;
err = PMSessionGetGraphicsContext(printSession,
        NULL, (void **)&printingPort );
```

- Mac OS X QuickDraw printing grafPort is unlike any on Mac OS 8/9
  - Hybrid QuickDraw/PostScript-like grafPort

How Printing Works for QuickDraw Applications

- Quartz 2D drawing can produce PDF output directly
- QuickDraw drawing during printing is translated to Quartz 2D drawing and therefore PDF
  - Pure QuickDraw applications
  - Apps mixing QuickDraw and Quartz-2D drawing must have QuickDraw portion translated

QD–PDF Translation



Process Is Similar to PostScript Printing

- Quartz 2D imaging model is essentially PostScript imaging model
- Borrowed techniques from the LaserWriter 8 driver
- Results consistent with LaserWriter 8 driver where possible

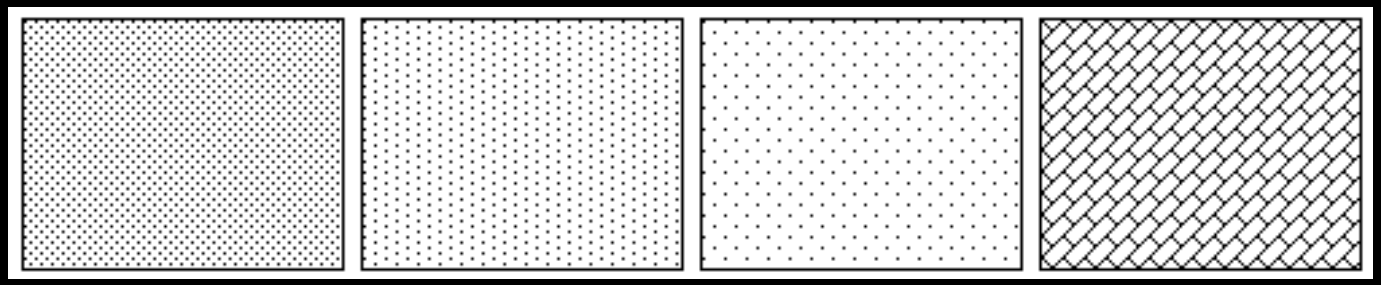
Similarities to LaserWriter 8

- Same general approach to translating QuickDraw to Quartz 2D imaging
- Same line layout algorithms
- Same QuickDraw transfer mode support
  - 1 bit deep source: all but srcXOr or notSrcXOr
  - Deep pixmaps: srcCopy
  - Coming soon: support for QuickDraw's "transparent" transfer mode

More Similarities to LaserWriter 8

- Same pattern substitution

Original QuickDraw Patterns



Substitute Patterns

Additional Similarities to LaserWriter 8

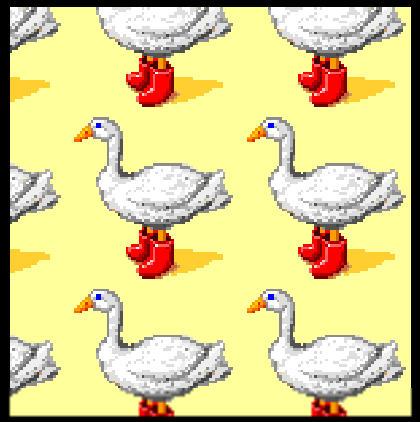
- PicComment support
  - Text alignment, rotation and flipping
  - Polygon smoothing
  - Graphics rotation
  - Dashed lines and fractional line width
  - ColorSync support

Differences From LaserWriter 8

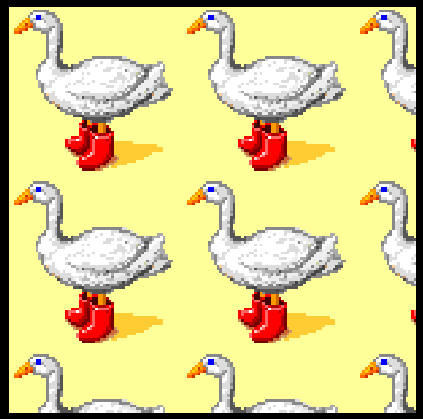
- PostScript picture comments are not supported
- PostScript PicComments are ignored
  - PDF has no PostScript interpreter
- QuickDraw drawing between PostScriptBegin/PostScriptEnd is imaged
  - Images QuickDraw representation of PostScript data
  - EPS preview image instead of EPS data

Improvements Compared to LaserWriter 8

- QuickDraw region support
  - Clipping, filling, and framing to arbitrary regions is supported
- QuickDraw pattern origin is respected



Incorrect



Correct

Demo

Mac OS X Printing grafPort Characteristics

Mixed QuickDraw and Quartz 2D Imaging During Printing

- CreateCGContextForPort does not work with printing port
  - During printing, QD to Quartz translator and app share same CGContext
  - To work properly requires careful state management of QuickDraw and Quartz drawing
- Working to resolve printing issues

Interleaving QD and Quartz Drawing (Future)

- QDBeginCGContext/QDEndCGContext (not in Mac OS X v10.0.x)
- QDBeginCGContext/QDEndCGContext routines work for both screen drawing and printing
- No QuickDraw imaging between QDBeginCGContext/QDEndCGContext
  - Port bottlenecks are replaced with non-drawing bottlenecks
- CGContext goes away with QDEndCGContext

Example:

```
CGContextRef ctx = NULL;
err = QDBeginCGContext(port, &ctx);
CGContextDrawPDFDocument(ctx,
                    bbox, pdfDoc, 1);
QDEndCGContext(port, &ctx); // ctx now
  NULL
CopyBits(…);
… other QD imaging
err = QDBeginCGContext(port, &ctx);
// ctx has initial gstate again
… more Quartz 2-d imaging
```

Linkable Routines Available for Mac OS X v10.0.x

- Allow applications to start BeginCG/EndCG style usage in Mac OS X 10.0.x
- Linkable routines do not enable printing
- Linkable routines do not install non-drawing bottlenecks between CGBegin/CGEnd
- Allow applications to adapt to the new semantics of CGBegin/CGEnd

Printing Tips

- Don’t expect PSBegin/PSEnd to generate PostScript
- Don’t expect PSBegin/PSEnd to hide QuickDraw drawing
- Apps that use StdText, StdBits, or other bottlenecks directly need to consult port bottlenecks
- Don’t expect a PixMap backing during printing (avoid ScrollRect!)

Documentation on Mac OS X Printing grafPort

- Inside Macintosh: Imaging With QuickDraw
  - See Appendix B:
    Using Picture Comments for Printing
- Coming soon:
  - Technote 2019:
    The Mac OS X Printing grafPort

Summary So Far…

- Can do cool stuff with Quartz 2D in Carbon on Mac OS X
- QD printing grafPort on X is hybrid QD/PS-like port
- Apps can mix QD/Quartz 2D content
  - QDBeginCGContext/QDEndCGContext
- What about making it go fast?

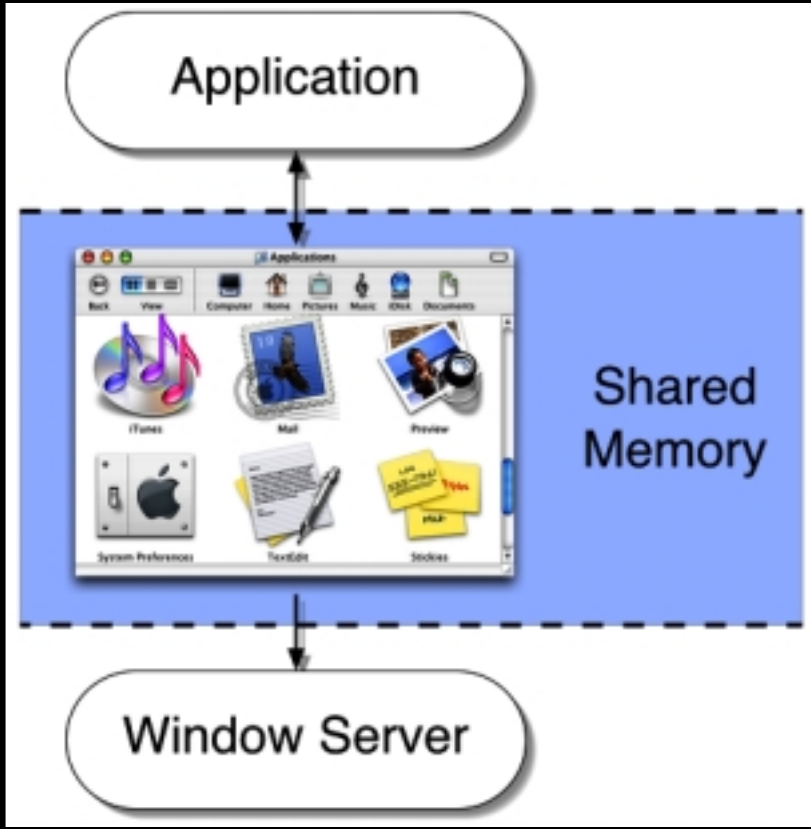
Quartz Performance Tuning

Ralph Brunner
Mineralogist

Quartz Performance Tuning

- Improve graphics speed
- About memory usage
- How to find graphics performance problems

About the Man Behind the Curtain…



Scratch My Backing Store, and I'll Scratch Yours!

- Enables system to mix content from different applications
- Window manipulation causes IPC
- Off-screen windows use memory
- Avoid re-sizing immediately after window creation (Cocoa)
- Avoid using off-screen GWorld to buffer window content (Carbon)

Drawing Attention

- Graphics speed is essentially a function of memory bandwidth
- Avoid multiple drawing
- Avoid unnecessary flushing
- Flushing is synced to the CRT beam

Traveling Window Buffers

- Don’t rely on window buffers living in main memory—backing stores may move to video memory in the future
- Avoid reading backing store bits directly (GetPortPixMap and GetPixBaseAddr)
- If you have to, we’d like to hear from you

Escape Velocity Engine

- Some graphics operations make use of the Velocity Engine
- Significant speed-up on G4 possible
- If you allocate bitmaps yourself, make rowBytes divisible by 16
- Possible pitfall: Test on G3, as performance may vary significantly

Demo

Ralph Brunner
Mineralogist

Roadmap

FF118 Feedback Forum: Graphics
Give us your feedback on Mac OS X graphics

Room J1
Fri., 2:00pm

Who to Contact

Travis W. Brown
Mac OS X Imaging Technology Manager
Apple Worldwide Developer Relations
travisb@apple.com

http://developer.apple.com/events/urls.html

Q&A

Travis W. Brown
Mac OS X Imaging Technology Manager
travisb@apple.com

http://developer.apple.com/events/urls.html

WWDC
Worldwide Developers Conference 2001